Improved Policies for Drowsy Caches in Embedded Processors

Junpei Zushi Gang Zeng Hiroyuki Tomiyama Hiroaki Takada
Graduate School of Information Science, Nagoya University
Furou-Cho, Chikusa-Ku, Nagoya, Aichi, 464-8603, Japan
{junpeiz, sogo, tomiyama, hiro}@ertl.jp

Koji Inoue
Department of Informatics, Kyushu University
744 Motooka, Nishi-Ku, Fukuoka, Fukuoka, 819-0395, Japan
inoue@i.kyushu-u.ac.jp

Abstract

In the design of embedded systems, especially battery-powered systems, it is important to reduce energy consumption. Cache are now used not only in general-purpose processors but also in embedded processors. As feature sizes shrink, the leakage energy has contributed to a significant portion of total energy consumption. To reduce the leakage energy of cache, the Drowsy cache was proposed, in which the cache lines are periodically moved to the low-leakage mode without loss of its content. However, when a cache line in the low-leakage mode is accessed, one or more clock cycles are required to transition the cache line back to the normal mode before its content can be accessed. As a result, these penalty cycles may significantly degrade the cache performance, especially in embedded processors without out-of-order execution. In this paper, we propose four mode transition policies which aim at high energy reduction with the minimum performance degradation. We also compare our policies with existing policies in the context of embedded processors. Experimental results demonstrate the effectiveness of the proposed policies.

1 Introduction

Power consumption has become one of the major concerns in today's processor design. Reducing power consumption can extend battery lifetime of portable systems, decrease chip cooling costs, and as well as increase system reliability. In the state-of-the-art processors, cache sizes have grown steadily to fill the widening gap between main memory latency and CPU clock frequency. As a result, the power consumption in cache has accounted for larger percentages of total system power. Most of the power increase stems from exponential growth in leakage due to the shrunk feature sizes. For instance, in the 70nm process technology, it has been estimated that cache leakage energy constitutes up to 70% of total cache energy [3].

To reduce the leakage energy of cache, a number of methods have been proposed so far in the literature. A common feature of these techniques is to provide two operation modes for cache. That is, a high leakage (awake) mode and a low leakage (sleep) mode. While in the high leakage mode, the cache can be accessed as a normal cache, it cannot be accessed in the low leakage mode until it returns to the awake mode. Depending on specific implementations, the sleep mode can be classified into state-preserving [1, 2, 3] and non-state-preserving [4, 5, 6] sleep modes, in which the former is realized by gating the supply voltage to the SRAM cell and the latter is realized by using dynamic voltage scaling. Generally, in the non-state-preserving sleep mode, the leakage savings are more significant than that of the state-preserving sleep mode. However, the non-state-preserving sleep mode imposes more performance degradation than the state-preserving sleep mode due to the increase of cache misses. As a typical state-preserving technique, Drowsy cache has been proposed in [1, 2, 3], in which cache lines are put into a low leakage drowsy mode. However, the performance degradation still exists when a cache hits on the drowsy cache line. In this case, one or more cycles are needed to move the line to the awake mode before its content is accessed. In high-performance processors, this latency can be hidden to some extent by out-of-order execution together with non-blocking cache accesses. In embedded processors, however, the latency cannot be mitigated due to its in-order execution.

In this paper, we focus on the Drowsy cache in embedded processors. Our aim is to achieve high leakage reduction with the minimum performance degradation. To this end, we propose four policies for mode transition management. These policies take advantage of the temporal locality of cache accesses and try to minimize accesses to drowsy lines by combining the idea of way prediction [8]

0-7695-3110-5/08 $25.00 © 2008 IEEE
DOI 10.1109/DELTA.2008.70

with the Drowsy cache. Since the lines that most possibly are accessed in the next time window are kept in the awake mode by our policies, the performance degradation is relieved. The main contributions of this paper include: (1) we propose new policies for mode transition management in the Drowsy cache, and (2) we evaluate the effectiveness of the Drowsy cache for single-issue in-order execution processors which are typically used in embedded systems.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 provides an overview of previous work on mode transition policies in the Drowsy cache. Section 4 presents the proposed policies. Section 5 shows our experimental results. Finally, Section 6 concludes the paper.

2 Related Work

As mentioned above, techniques for cache leakage reduction can be classified into state-preserving and non-state-preserving technique. In this section we overview these existing techniques as this classification.

Yang et al. proposed non-state-preserving Dynamically ResIzable cache (DRI)[6] to reduce leakage of instruction cache in which the unnecessary cache lines are moved to low leakage mode by using power gating technique when cache miss rate is lower than a given threshold. Kaxiras et al. proposed non-state-preserving cache decay[4] to reduce leakage of data cache. In their approach, the cache lines that have not been accessed during a given decay interval will be moved to low leakage mode by using power gating technique. In many state-of-the-art processors, cache is constructed from more than two levels to further improve performance. Considering this cache hierarchy, Li et al. proposed leakage energy management to reduce the leakage energy of cache [5]. When cache miss occurred in upper level cache, conventional cache copies data from lower level cache to upper level. After that, lower level cache does not need to keep the data which has been copied to upper level cache anymore. Therefore, in this case, the approach puts lower level cache line into low leakage mode by using power gating technique.

In contrast to the above non-state-preserving low leakage techniques, a state-preserving leakage reduction technique, called Drowsy Cache, has been proposed in [1, 2, 3]. In this method, the cache lines are moved to a low leakage drowsy mode by lowering it supply voltage. Because the contents of cache line have been preserved, the performance degradation caused by cache miss increase in non-state-preserving methods can be avoided. However, when a cache hit on a drowsy mode line, one or more clock cycles are required to transition it back to awake mode before its contents can be accessed, thus performance degradation still exists in this case. Generally, more hits on the drowsy cache lines, more performance degradations may be expended. To tackle this problem, Petit et al. improved the original drowsy cache in [9]. The detailed description of the original drowsy cache and improved one will be given in the following section.

Thinking of the temporal locality of the memory accesses, the method of [8] predict the cache way which will be access near the future by using MRU(Most Recently Used) algorithm. This method is used to reduce dynamic energy of cache. When the cache is accessed, conventional cache compare the cache tag for all cache way. But in the way prediction cache approach, only the cache tag of the MRU way is compared first. By using way prediction, the number of comparing cache tags can be reduced, thus the dynamic energy can be reduced too.

3 Previous Policies

3.1 Simple / Noaccess Policies [2]

Two mode change policies have been proposed to reduce leakage energy in [2]. These methods reduce leakage energy by changing cache lines into low leakage mode periodically.

1. Simple Policy
   Transition all cache lines into low leakage mode at regular time window.
2. Noaccess Policy
   Transition cache lines into low leakage mode that have not been accessed in the previous time window.

Generally, there are two-fold factors should be considered when determining a policy for controlling mode transition. On one hand, the more lines stay in the low-leakage mode, the more leakage can be reduced. On the other hand, keeping as many as possible lines in the low-leakage mode may result in more mode transitions which means more performance degradation. Therefore, a reasonable policy should find a good balance between the leakage reduction and performance degradation. As for the above policies, "Simple policy" increases the number of low-leakage cache lines, but mode transition occurs frequently and performance degradation is significant. In contrast, "Noaccess policy" reduces the number of mode transitions at the cost of reduced number of low-leakage cache lines which means less leakage savings. Thus, more effective policy is required to improve the original drowsy cache policies.

3.2 MRO / TMRO / RMRO Policies [9]

To tackle the above problem, three other policies for controlling mode transition in drowsy cache have been proposed in [9]. Differ from the original policies, the temporal locality of memory access is exploited in these policies.

1. MRO (Most Recently used On) Policy
   This method transitions all cache lines to low leakage mode except the MRU line in each cache set. Therefore, only one line in a cache set is always in awake mode.
2. TMRO (Two Most Recently used On) Policy
   This policy keeps two MRU lines as active in each

cache set. Therefore, two lines in each cache set are always in awake mode.

3. RMRO (Reused Most Recently used On) Policy
   Similar to the original drowsy cache policies, this policy also utilizes regular time window to control mode transition of cache line. Specifically, it chooses lines to keep awake as the following process:
   - If no line was accessed during the previous time window , maintain the whole line in low leakage mode.
   - If only one line was accessed on the previous time window, keep awake only the most recently used line.
   - If more than one line was accessed on the previous time window, keep awake only the two most recently used lines in the set.

MRO and TMRO policies improved the original policies by exploiting temporal locality. The performance degradation can be relieved by keeping the one or two MRU lines always active. However, keeping the MRU always active even if it has not been accessed for a long time, will waste the potential for leakage savings. In contrast, the RMRO exploited both temporal locality and access history information in previous time windows to find a better balance between leakage reduction and performance degradation. Although the problem of performance degradation in original drowsy cache has been solved to some extent, it still exists due to the limited number of awake lines, i.e., most two lines can be awake anytime. Thus, when a cache hit on any lines except the two MRU lines, the penalty cycles will be occurred, leading to performance degradation.

4 Improved Policies

4.1 Overview

Considering the above reasons, we propose four policies for controlling the mode transition in drowsy cache. These policies are proposed to find a more precise balance between leakage reduction and performance degradation, in particular to avoid performance degradation but still achieve significant leakage reduction. These policies have some common features as follows.

- All policies utilize time window for controlling mode transition of cache line at fixed period
- All policies utilize MRU information to predict the possible access lines in each cache set in the next time window
- All policies only limit the number of awake lines in each set at the beginning of each time windows but not limit the maximum number of awake lines during the entire time window.
- AAM and AOM utilize history access information in the previous time window to predict the possible access lines in the next time window.

4.2 Proposed Policies

1. PMRO (Periodic MRO) Policy
   In this policy, all cache lines are put into low leakage mode at fixed window period except for the MRU line in each cache set. The line in low leakage mode will be woken up when it is accessed.
2. PTMRO (Periodic TMRO) Policy
   This policy is similar to the PMRO except that it leaves two MRU lines in each cache set in awake mode, and puts other lines into low leakage mode.
3. AAM (Accessed And MRU) Policy
   In this policy, all cache lines are put into low leakage mode except for the MRU line that has been accessed in the previous time window. It means that if no access occurs in the previous window, then all lines will be put into low leakage mode. By this constraint, we expect to achieve more leakage savings in this policy.
4. AOM (Accessed Or MRU) Policy
   This policy put all cache lines into low leakage mode except for the MRU line and the accessed lines in the previous time window. The difference with AAM is that at least the MRU line will be kept awake anytime.

It is important to note that once the MRU information has been memorized it will hold and cannot be eliminated even if no access occurs in one or more continuous time windows. The PMRO and PTMRO are improved version of MRO and TMRO by incorporating periodic time window for mode transition control. Moreover, while MRO and TMRO limit the maximal number of awake lines to one and two at anytime, PMRO and PTMRO eliminate this limitation. Therefore, they are expected to achieve improved performance. Actually, the AAM is similar to the PMRO only with difference in the case of no access in the previous window. In this case, PMRO keeps MRU awake, but AAM put all lines into low leakage mode. Table 1 summarizes the differences between these policies.

Note that the proposed policies do not require additional hardware to implement them comparing with previous policies. The descriptions about detailed hardware implementation of policies can be found in [2, 9].

5 Evaluation

5.1 Energy Calculation

We utilize the leakage model as used in [7] to estimate the leakage energy (LE_{total}) of L1 data cache.

$$LE_{total} = CC \times LE_{line} \times CSize \quad (1)$$

$$CC = CC_{conv} + CC_{extra} \quad (2)$$

$$LE_{line} = SR \times LE_{sline} + (1 - SR) \times LE_{aline} \quad (3)$$

where CC is the total number of execution cycles, LE_{line} is the average leakage energy consumed in a cache line per cycle, and $CSize$ is the total lines in the L1 data cache. CC

Table 1. Comparison of policies

Policies	Use of MRU	Use of access history	Use of TW	Min num of awake lines per set	Max num of awake lines per set	Num of awake lines at the beginning of a TW	Source
Simple	N	N	Y	0	num of ways	0	[1, 2]
Noaccess	N	Y	Y	0	num of ways	0 – num of ways	[1, 2]
MRO	Y	N	N	1	1	always 1	[9]
TMRO	Y	N	N	2	2	always 2	[9]
RMRO	Y	N	Y	0	num of ways	0 – 2	[9]
PMRO	Y	N	Y	1	num of ways	1	this work
PTMRO	Y	N	Y	2	num of ways	2	this work
AAM	Y	Y	Y	0	num of ways	0 – 1	this work
AOM	Y	Y	Y	1	num of ways	1 – num of ways	this work

consists of CC_{conv} and CC_{extra}. CC_{conv} is the conventional execution time without any leakage reduction techniques, while CC_{extra} is the penalty time caused by accessing to the lines in the low leakage mode. LE_{line} is determined by SR which is a rate of low leakage mode lines in the entire cache. LE_{sline} and LE_{aline} are the cache-line average leakage energy dissipated in one cycle on the low leakage and the awake mode, respectively.

We utilize the following equation to estimate the total energy overhead for mode transitions.

$$CE_{total} = CE_{line} \times MCTimes \quad (4)$$

where CE_{line} is the energy consumption for transitioning a cache line from low leakage mode to awake mode, and $MCTimes$ is the total number of mode transitions.

We thus calculate the total energy consumption by adding energy overhead to the total leakage energy.

$$Energy = LE_{total} + CE_{total} \quad (5)$$

Furthermore, for the purpose of comparison of different policies, we calculate an energy-delay product (EDP) using the following equation.

$$EDP = CC \times Energy \quad (6)$$

We use the parameters as shown in Table 2 for emperiments which are translated from the original data as presented in [2]. Note that we also omit the energy consumption of main memory as done in previous evaluations [2, 9]. Because the application of drowsy cache does not change the cache miss rate, the dynamic energy of main memory will maintain the same as normal cache.

Table 2. Energy parameters

LE_{aline}	LE_{sline}	CE_{line}
4.17E-13 (J)	6.63E-14 (J)	2.56E-11 (J)

5.2 Experimental Environment

To evaluate previous policies and our proposed policies, as presented in Sections 3 and 4, an architectural simulator SimpleScalar/ARM[10, 11] is employed to output memory access trace. The generated traces are fed to our Drowsy cache simulator in which all nine kinds of policies for mode transition have been implemented. The configuration of "SimpleScalar/ARM" is summarized in Table 3. Note that because a 4096 window length reaches a reasonable compromise between performance and energy savings, we take this value for our experiments. Several benchmarks from MediaBench[12] are used in the experiments.

Table 3. Cache configulation

Parameters	Value
Cache	Data L1 cache
Cache size	16KB / 32KB
Cache line size	32B
Associative	2 / 4 / 8 ways
Cache miss penalty	20 cycles
Mode change penalty	3 cycles
Time window cycles	4096 cycles

Taking account of the complexity of circuit implementation, we classify all these policies into two groups according to whether access history information is used or not. One group includes Simple, MRO, RMRO, PMRO, and PTMRO policies which do not use access history for determination. In contrast, another group includes Noaccess, RMRO, AAM, and AOM policies which use access history. Because the former group does not use the history of access, they can be implemented in simple hardware than the latter group.

5.3 Results for Policies not Using Access History

In this section, we compared the policies which do not use the information of access history in past time windows. Figure 1 shows the results of individual benchmark programs for a 32KB, 4-way set-associative cache, and Figures 2 and 3 show the average results over all benchmarks for 16KB and 32KB caches, respectively. Each value is normalized with respect to conventional cache.

As Figure 1 shows, the Simple policy achieves the best energy-delay product for four of the eight programs, while the PMRO policy is the best for the other programs. On the average, as Figures 2 and 3 show, the Simple policy is the best for the 2-way associative cache, while the PMRO is the best for higher associative caches.

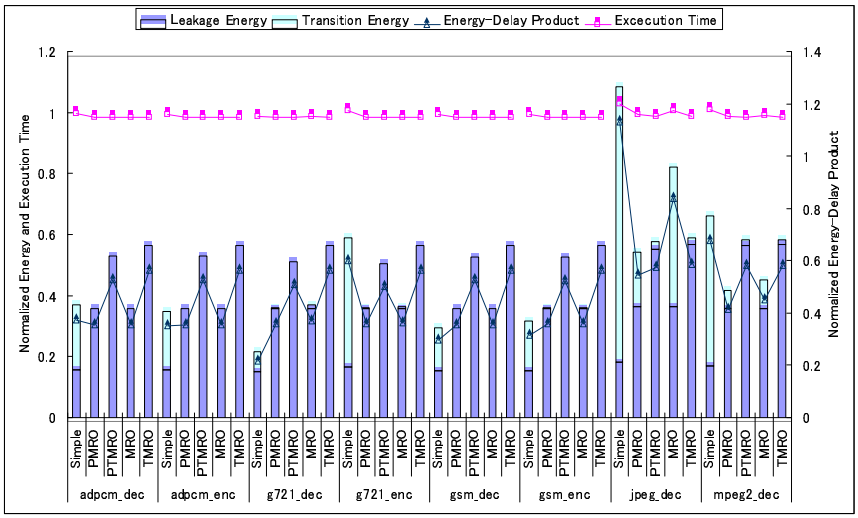

Figure 1. Energy-delay product of benchmarks for policies not using access history (32KB 4-way)

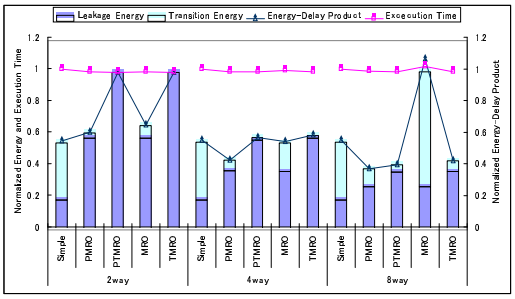

Figure 2. Average energy-delay product for policies not using access history (16KB)

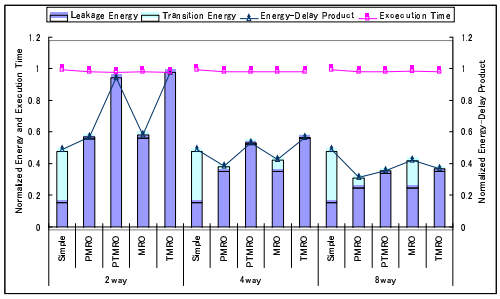

Figure 3. Average energy-delay product for policies not using access history (32KB)

The results also show that the performance degradation due to the mode transition overhead is not significant even for single-issue in-order processors. This indicates that the Drowsy cache is effective not only for high-end processors but also for embedded processors.

We had expected that PMRO would give a good trade-off between performance and energy reduction, but that was not confirmed in our experiments. Instead, our results show that PMRO gives a good balance between leakage energy and mode transition energy, which leads to an overall energy reduction.

5.4 Results for Policies Using Access History

As a next set of experiments, we compared the policies which use the information of access history in the past time windows. The results are shown in Figures 4, 5 and 6. In these experiments, AAM and AOM, both of which are proposed in this paper, cannot improve the energy reduction for most cases. Only for adpcm_dec and jpeg_dec which feature rich memory-access locality, AOM gives the best energy-delay product.

The energy-delay product of the policies which uses access history is generally better than that of ones which do not use it. However, it should be noted that the two groups cannot be compared directly because the complexities of their hardware implementation are different.

6 Conclusions

In this paper, we have proposed four policies for mode transition management to achieve high leakage reduction with minimum performance degradation. We have also evaluated five previous policies and the proposed ones on a single-issue in-order embedded processor. According to the hardware implementation of these policies, we evaluated the nine policies as two groups. Experimental results reveal the followings.

- Previous methods and the proposed ones can still work well on the in-order embedded processor without imposing large performance degradation
- All policies have closed performance degradation on the embedded processor, thus the dominant factor of energy-delay product is the energy consumption.
- For policies that do not use the access history, our approach PMRO achieves the best energy-delay-product than previous methods. Meanwhile, for policies that use access history, the Noaccess policy achieves relatively better results than others.
- The results on individual benchmark programs show that the best policies differs with the programs.

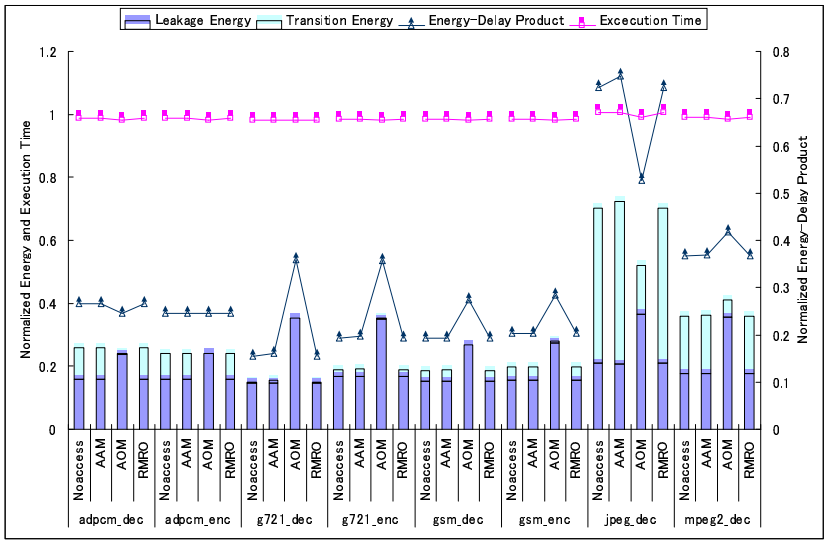

Figure 4. Energy-delay product of benchmarks for policies using access history (32KB 4-way)

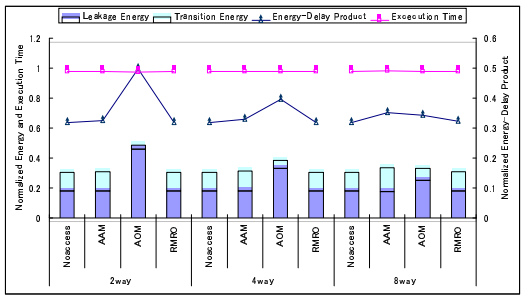

Figure 5. Average energy-delay product for policies using access history (16KB)

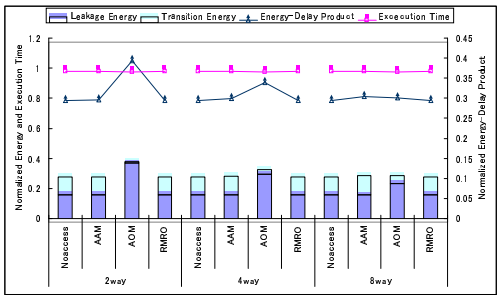

Figure 6. Average energy-delay product for policies using access history (32KB)

Because an embedded system is optimized for a specific application, it is important to select the optimal policy for different applications. Our future work will exploit this application-depend memory access feature to find the optimal policy for a specific embedded system.

7 Acknowledgments

We thank Dr. Honda Shinya, Reiko Komiya and Hideki Takase for helping our experiments. We also thank Dr. Tetsuo Yokoyama for suggesting our experiments. This work is supported in part by the Core Research for Evolutional Science and Technology (CREST) from Japan Science and Technology Agency.

References

[1] N.S. Kim, K. Flautner, D. Blaauw, T. Mudge, "Circuit and Microarchitectural Techniques for Reducing Cache Leakage Power," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 12, no.2, 2004.

[2] K. Flautner, N.S. Kim, S. Martin, D. Blaauw, T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," *International Symposium on Computer Architecture*, 2002.

[3] N.S. Kim, K. Flautner, D. Blaauw, T. Mudge, "Drowsy Instruction Caches: Leakage Power Reduction Using Dynamic Voltage Scaling and Cache Sub-bank Prediction," *International Symposium on Microarchitecture*, 2002.

[4] S. Kaxiras, Z. Hu, M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," *International Symposium on Computer Architecture*, 2001.

[5] L. Li, I. Kadayif, Y-F. Tsai, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, A. Sivasubramaniam, "Leakage Energy Management in Cache Hierarchies," *International Conference on Parallel Architectures and Compilation Techniques*, 2002.

[6] S.H. Yang, M.D. Powell, B. Falsafi, K. Roy, T.N. Vijaykumar, "An Integrated Circuit/Architecture Approach to Reducing Leakage in Deep-Submicron High-Performance I-Caches." *International Symposium on High-Performance Computer Architecture*, 2001.

[7] R. Komiya, K. Inoue, V.G. Moshnyaga, K. Murakami, "Quantitative Evaluation of State-Preserving Leakage Reduction Algorithm for L1 Data Caches," *IEICE Trans. Fundamentals*, vol. E88-A, no. 4, 2005.

[8] K. Inoue, T. Ishihara, and K. Murakami, "Way-Predicting Set-Associative Cache for High Performance and Low Energy Consumption," *International Symposium on Low Power Electronics and Design*, 1999.

[9] S. Petit, J. Sahuquillo, J.M. Such, D. Kaeli, "Exploiting Temporal Locality in Drowsy Cache Policies," *Computing Frontiers*, 2005.

[10] D. Burger, T. M. Austin, "The SimpleScalar Tool Set Version 2.0," *UW Madison Computer Sciences Technical Report #1342*, June. 1997.

[11] *http://www.simplescalar.com/*

[12] C. Lee, M. Potkonjak, W.H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems," *International Symposium on Microarchitecture*, 1997.